

Innovative Integration

LabVIEW driver for ADC64 User's Manual

The *LabVIEW driver for ADC64 User's Manual* was prepared by the technical staff of Innovative Integration, October 1998. It describes Version 1.00 of the software.

For further assistance contact

Innovative Integration
5785 Lindero Canyon Road
Westlake Village, CA 91362

PHONE (818) 865-6150
FAX: (818) 879-1770
email: techsprt@innovative-dsp.com
FTP: ftp.innovative-dsp.com
WWW: http://www.innovative-dsp.com

This document is Copyright 1998 by Innovative Integration. All rights are reserved.

Document: VSS\Documents\Manuals\LabVIEW Drivers\ADC64\LabVIEW_ADC64_UM.doc

Table of Contents

1. Introduction	5
2. Installation	5
2.1 Target Hardware Requirements	5
2.2 Host Hardware Requirements	6
2.3 Software Installation	6
2.4 Distribution Files	6
2.5 Multiple Target Board Support	7
3. The LabVIEW driver	8
3.1 Overview	8
3.2 Features	8
3.3 Basic Board Operation	9
3.4 Recording Mode Operation	9
3.4.1 Limitations	9
3.4.2 Programming Notes	9
3.5 Playback Mode Operation	11
3.5.1 Buffer Data Format	11
3.5.2 Programming Notes	11
4. LabVIEW Driver Functions by Category	12
4.1 Initialization Functions	12
4.2 Command Functions	12
4.2.1 Target Configuration	12
4.2.2 Recording Mode	12
4.2.3 Playback Mode	12
4.2.4 Asynchronous Commands	13
4.3 Status Functions	13
4.4 Buffer Pool Functions	14
4.4.1 Pool Control	14
5. Example Programs	15
5.1 Scope	15
5.1.1 Running the Scope Demo	15
5.1.2 Inside the Scope Demo	15
5.2 SigGen	16
5.2.1 Running the SigGen Demo	16
5.2.2 Inside the SigGen Demo	16
6. LabVIEW Driver VIs	17

1. Introduction

Innovative Integration's Supercontrollers have the DSP power to be impressive engines for data acquisition, but programming the DSP to obtain maximum data throughput is not an easy task. It requires learning a second tool set, a second compiler, and a second assembly language for extremely time critical code.

Yet for many projects a turnkey system is not acceptable either. Often the data acquisition needs to be a part of a larger system with its own user interface. In other cases, additional functions or signals may need to be emitted before, during, or after data taking.

The LabVIEW driver for the ADC64 addresses these concerns by providing a system that streams data from the ADCs or sends data to the DACs at up to 150 kHz on the ADC64. These streaming modes can be fully set up, configured, and initiated from the host under LabVIEW, without any target programming on the DSP required. Additional VI calls are used to insert or extract data blocks from an internally managed buffer pool to the LabVIEW environment.

In addition to the streaming functions, the LabVIEW driver also provides a set of VIs that give full access to the analog and digital I/O hardware of the target board asynchronously. Although these VIs will not support the data rates achievable by the streaming I/O, they can perform their I/O in **parallel** with the streaming modes, allowing low rate data channels to be read or digital signals to be fired even during a streaming mode recording or playback session. In addition, they can provide a simpler interface useful for low rate applications such as calibration and board functionality testing.

All of the above features require no target programming.

All the functionality of the LabVIEW driver is implemented using the Ventura DLL for the ADC64. If not using LabVIEW, you can make calls to Ventura directly or if need data transfer capability with additional custom target processing can use OpenVentura. Contact Innovative Integration for more information on Ventura.

2. Installation

Installation of the LabVIEW driver consists of both hardware and software installation procedures. The included *Development Package Installation Guide* document describes the processes required to install the software.

This document is intended to augment, not replace, the *Installation Guide*.

2.1 Target Hardware Requirements

The current release of LabVIEW driver supports only the Innovative Integration (II) ADC64 PCI Bus DSP Data Acquisition Card or the cADC64 CompactPCI DSP Data Acquisition Card as the target coprocessor. The ADC64 is a PCI Bus compatible DSP data acquisition card incorporating a Texas Instruments TMS320C32 floating point DSP processor integrated with 16 bit analog input/output capability, 16 bits of digital I/O, and up to 128K words (512K bytes) of onboard SRAM. The cADC64 has an identical feature set and is compatible with the CompactPCI bus.

The ADC64 must be Revision F or later to function with Ventura. It must have the 128K words of memory. If multiplexing of input channels is ever used, then the Anti-Alias filters on the ADC inputs **must** be modified or else the signals on different channels will be mixed together.

Additional information on the ADC64 hardware, including connector pinouts, jumper configuration and filter schematics can be found in the *ADC64 Hardware Manual*, available with the board itself.

For brevity, the ADC64 and cADC64 will be referred to in the rest of this manual collectively as the ADC64.

2.2 Host Hardware Requirements

The software development tools for the LabVIEW driver software require an IBM or 100% compatible Pentium class or higher machine for proper operation. The host must have at least 16 Mbytes of memory and a free PCI slot. Windows 95 or NT (referred to herein simply as *Windows*) is required to run the software.

2.3 Software Installation

The LabVIEW driver is installed as a final step in installing the hardware and software as described in the *Installation Supplement*. In summary, the following items need to be loaded first:

- 1) The ADC64 Device Driver or VxD software.
- 2) The ADC64 hardware.

Note: the order of installation of these packages is important. Do not attempt to install the target DSP hardware until the software installation is complete.

2.4 Distribution Files

The LabVIEW driver Distribution consists of many files in a number of directories.

File or Directory	Purpose
labview\Scope.vi	Scope Example demonstrating Record mode.
labview\SigGen.vi	SigGen Example demonstrating Playback mode.
Lib\Host	Host DLLs
...\VenAdc64.dll	Ventura DLL
...\ven2LV.dll	LabVIEW interface DLL

Warning: Can't have more than one version of Ventura DLL or LabVIEW interface DLL or basic board DLL (adc64dll.dll) on your system because LabVIEW VIs could link with the different versions and the Ventura environment will be corrupted.

2.5 Multiple Target Board Support

The LabVIEW driver for the ADC64 currently does not support the simultaneous use of more than one ADC64 board. The target ADC64 to use can be configured at the start of the program and so multiple boards can be used on successive runs of the program without difficulty.

3. The LabVIEW driver

3.1 Overview

The LabVIEW driver is a software layer that provides a high level interface to the ADC64 target board to allow it be used as data acquisition board under LabVIEW. Target level programming is not required, since a standard target COFF file is provided that can perform all the functionality needed by the functions of the LabVIEW driver. This COFF file is downloaded to the target in ADC64_Open.vi

The LabVIEW driver uses the Ventura DLL which implements a message passing protocol to allow commands to be passed from target to host. There are many predefined messages to access board functions. In addition, data can be streamed from the ADCs on the target to the host (Recording Mode) or from the host to the target DACs (Playback Mode). These modes are high performance, allowing I/O at 150 kHz per channel, but require special configuration and buffer management functions.

The Ventura DLL uses as its base the functions in the standard board DLL provided with the ADC64 card.

3.2 Features

- **Recording Mode:** A streaming mode that allows data acquisition from 1 to all 8 ADCs of the ADC64 at rates up to 150kHz. Each ADC can be multiplexed up to 8 times giving a total of 64 channels of input at up to 25 kHz per channel. Gains for each ADC may be independently set at the start of the recording. Data is delivered to the application program in a packed format on the host as it is obtained.
- **External Start Trigger:** The start of a Recording session may be triggered by an external hardware signal or by command from the application
- **Playback Mode:** A streaming mode that allows up to 2 channels to be played back on the 2 DACs of the ADC64. The configuration allows these channels to be selected out of a larger Recording run, including one with multiplexing. Output rates of up to 200 kHz are supported.
- **External Triggering:** All modes may be optionally triggered by an external clock source.
- **Data Buffer Management:** Data buffers for streaming modes are managed internally by the LabVIEW interface DLL. The user is presented with VIs that allow sending data to Ventura or receiving data from Ventura. Similarly, using the playback VIs allow preloading of output data before starting a Playback Mode stream.
- **Target Queue Status:** A function can be used to find the depth of the two data queues on the target, the DAC Queue and the ADC Queue.
- **Digital I/O Support:** Functions are provided to asynchronously configure, read, or write the 16 bits of digital I/O on the ADC64
- **Asynchronous Analog I/O Support:** Functions are provided to asynchronously read from any of the 8 ADCs on the ADC64, write to each of the 2 DACs on the board, and to set the ADC gains and the Mux channel.

3.3 Basic Board Operation

The simplest way to use the LabVIEW driver to operate the ADC64 is to use only the Asynchronous Command functions in the API. These functions can access all of the analog and digital features of the board and do not require any setup or configuration. The disadvantage is that these functions only can obtain data at a limited rate. This rate is about 2500 operations per second in the absence of other processing on the target. While this rate is not very useful for high rate data acquisition, it is suitable for use in simple board functionality and calibration programs.

See the section on LabVIEW Driver VIs for more information on the Asynchronous commands.

3.4 Recording Mode Operation

Record mode is the typical mode for LabVIEW driver operation, because the ADC64 board is best suited for Analog Input. Before starting the run, a pool of pre-allocated buffers is created in host memory. Data will be transferred from the target into this pool, where the application can extract and process it.

The ADC64 is then configured to take data at a certain rate from some number of ADCs. Multiplexing to any depth is supported, and the gains for each ADC can be individually set. When this mode is started, the ADCs are read and placed in a data queue. The format of these events is different for different configurations, but is well defined.

Once enough data is accumulated to fill a pool buffer, it is extracted from the data queue on the target and transferred to the host by a busmaster transfer, and the Ventura DLL is signaled that a new data buffer is present. This data is transferred from busmaster memory into the pool, freeing that memory for future busmaster transfers. Functions are provided to the user to detect that new data was entered into the pool and to get access to the data and release its buffer for reuse.

See the section on LabVIEW driver VIs for more information on the Record Mode commands.

3.4.1 Limitations

Record mode operation is subject to the following limitations:

- The ADCs must be selected starting from ADC0 and upward from there. That is, a 4 ADC run is always ADC 0, ADC 1, ADC 2 and ADC 3.
- Multiplexing always starts at setting 0 and moves upward from there. That is, a run with a mux depth of 2 uses mux channels 0 and 1.
- An explicit Stop message must be sent to cease recording.

3.4.2 Programming Notes

3.4.2.1 Opening and Closing LabVIEW driver

The first step in programming a record mode application is to open the Ventura DLL with a call to *ADC64_Open.vi*. This opens the target board and downloads a COFF file to it. The LabVIEW driver uses the Ventura.out file included with the distribution.

The final step in an application should be to call *ADC64_Close()* to shut down the DLL and free memory.

3.4.2.2 Buffer Pool Management

Buffer pool management is handled for the user by several VIs. *ADC64_Open.vi* sets up the buffer pool and *ADC64_Close.vi* releases pool resources.

3.4.2.3 Configuration & Starting

To start each run, a *ADC64_ConfigAcq.vi* must be called to enter the recording mode. Once configured, starting the run is a two-stage process. First call *StartRecordMode.vi*, which starts the analog interrupt but does not start data taking.

The run will continue until stopped by command from the user.

To start data taking, the application has two choices. Either the *TriggerRecordMode.vi* function can be called by the application, or a signal on an external Start Trigger line can be used to synchronize the acquisition with an external event.

To stop a recording run, call *AbortRecordMode.vi*.

3.4.2.4 External Triggering

To use external triggering, set the External Trigger word in the configuration structure to 1, and wire a clock input to the EXT_TRIG inputs for the ADC pairs your configuration wishes to trigger. See the ADC64 Hardware Manual for details on the type of signal and which pins to connect to.

3.4.2.5 External Start Triggering

The hardware signal for the External Start Trigger uses the TCLK0 pin on the ADC64 external connector P1, pin 52. A high to low edge will start data recording

3.5 Playback Mode Operation

Playback mode was intended to allow a subset of a stored recording to be played back on the DACs of the ADC64. Due to interface restrictions of LabVIEW this direct correspondence to a recording has been eliminated and replaced with a simpler direct interface.

On the target, the ADC64 first attempts to request enough data to prefill all of its internal buffers. If there is no data initially available, the queue will be left empty, which might cause dropouts of data during the beginning of a run. For this reason it is best to prefill as much data as possible to assure that data can be fed as fast as it is needed from the very start of the run. Once this is done, the data output begins. Whenever the target finds it has enough room in its internal buffers to accept a new pool buffer the Ventura DLL is signaled with a request for new data. This data is transferred into busmaster memory from the pool, and the buffer in the pool is released for reuse. Ventura then signals the target that the data is available, and it transfers the data into its target data queue.

Functions are provided to the user to detect when buffers in the pool are emptied and released for reuse. The user application can then insert the next buffer into the pool for processing. All the previous processing is handled internally by the driver and/or Ventura DLL so the user doesn't need to worry about it.

3.5.1 Buffer Data Format

Playback mode assumes the same event format as the Record mode does. Because of this data that has been previously recorded can be output as playback without any data reformatting. If generating a new data set, it is simple enough to insert the data into a buffer into the proper format for a 1 or 2 channel run.

3.5.2 Programming Notes

3.5.2.1 Opening and Closing Ventura

For Playback mode, the Open and Close functions should be called just as in Record Mode.

3.5.2.2 Buffer Pool Management

For Playback mode, the Buffer Pool functions should be called just as in Record Mode.

3.5.2.3 Configuration & Starting

To start each run, *ADC64_ConfigPlay.vi* must be called to enter the mode. Its argument is a data structure that is to be filled in with the parameters of the run. See the Ventura Data Structures section for the layout of this structure. Once configured, call *StartRecordMode.vi* to begin the playback.

To stop a playback run, call *AbortPlaybackMode.vi*.

3.5.2.4 External Triggering

To use external triggering, set the External Trigger word in the configuration structure to 1, and wire a clock input to the EXT_TRIG5 input for the DAC pair. See the ADC64 Hardware Manual for details on the type of signal and which pins to connect to.

4. LabVIEW Driver Functions by Category

The following is a summary listing of functions in the LabVIEW driver by category of use. There is an additional listing of the LabVIEW driver by function in the back of this manual which has additional details on the use of the each VI.

4.1 Initialization Functions

ADC64_Open.vi

ADC64_Close.vi

ADC64_Open.vi is the general initialization functions for the driver. It opens the target board and downloads a program to the target and prepares it for use by the other functions. *ADC64_Close.vi* cleans up and closes the resources used by the driver when it is finished.

4.2 Command Functions

4.2.1 Target Configuration

ConfigureBoard.vi

ConfigureBoard.vi sends data to the board containing any information about the configuration of the board that cannot be determined by the target itself, and it sends the size of the data buffers being transferred from the target to host. This command must be sent before any other command to ensure that the target side code can properly coordinate streaming mode accesses with the Ventura DLL.

4.2.2 Recording Mode

ADC64_ConfigAcq.vi

StartRecordMode.vi

TriggerRecordMode.vi

AbortRecordMode.vi

AdcBufferAvailable.vi

GrabAdcBuffer.vi

These six commands control the Recording Mode streaming. *ADC64_ConfigAcq.vi* sends a description of the session to the target, including the channels to be acquired, how long to acquire data, the ADC gains, and if multiplexing is to be used. *StartRecordMode.vi* begins the data taking, but output is not enabled until either an external start signal or the *TriggerRecordMode.vi* function is called. *AbortRecordMode.vi* will cause the Recording session to be terminated prematurely.

The Buffer Pool is set up internally by the LabVIEW driver and streaming is accomplished with *AdcBufferAvailable.vi* and *GrabAdcBuffer.vi*. Data must be extracted from the pool to provide free buffers in the pool for streamed data, or data will be lost.

4.2.3 Playback Mode

ADC64_ConfigPlay.vi

StartPlaybackMode.vi

AbortPlaybackMode.vi

DacBufferAvailable.vi

GrabDacBuffer.vi

These five commands control the Playback Mode streaming. *ADC64_ConfigPlay.vi* sends a description of the session to the target, including the channels to be output, how long the data file is, the structure of the input buffer, and the data rate. *StartPlaybackMode.vi* begins the data output. *AbortPlaybackMode.vi* will cause the Playback session to be terminated prematurely.

The Buffer Pool is set up internally by the LabVIEW driver and controlled by *DacBufferAvailable.vi* and *GrabDacBuffer.vi*. Data must be entered into the pool to provide full buffers in the pool for streamed data, to supply the output to the DACs.

4.2.4 Asynchronous Commands

4.2.4.1 Digital I/O Functions

DioDirection.vi

DioWrite.vi

DioWriteBit.vi

DioRead.vi

DioDirection.vi configures the bytes of digital I/O for output byte by byte. *DioWrite.vi* and *DioWriteBit.vi* write data out the digital output port; *DioWrite.vi* updates all bits at once while *DioWriteBit.vi* updates a single bit. *DioRead.vi* reads data from digital input; it reads all bits at once.

4.2.4.2 Analog I/O Functions

DacWrite.vi

AdcRead.vi

DacWrite.vi updates a single DAC channel with a new value. *AdcRead.vi* performs a conversion on the ADC and returns the value to the host.

SetMux.vi

GetMux.vi

These functions allow the setting and reading back of the multiplexer channel for an ADC.

SetGain.vi

GetGain.vi

These functions allow the setting and reading back of the Gain setting for an ADC. The setting is a “gain mode” index from 0 to 3, to support both binary (x1, x2, x4, x8) and decimal (x1, x2, x5, x10) gain amplifiers without change.

4.3 Status Functions

TargetQueueStatus.vi

TargetQueueStatus.vi informs the application of the percent full value of the target-side data queues. The argument selects which queue, the ADC queue or the DAC queue.

4.4 Buffer Pool Functions

4.4.1 Pool Control

PoolStop.vi

PoolReset.vi

PoolStop.vi is used when stopping data acquisition, as it sends a signal that ends all waiting for free or empty buffers. *PoolReset.vi* resets the pool internal information to start a new run. Its single argument tells which type of streaming is to be used, Recording or Playback.

5. Example Programs

The LabVIEW driver for the ADC64 includes two sample applications. Each illustrates a single aspect of using the driver. The Scope example uses Recording Mode to display input channels. The SigGen example is a simple signal generator on the two DACs and illustrates using the Playback Mode.

5.1 Scope

5.1.1 Running the Scope Demo

The Scope.vi example displays the signals input on IN0, IN8, IN16, IN24, IN32, IN40, IN48 and IN56 of the ADC64.

Set desired MuxDepth, AdcCount and DataRate and then press Run button on LabVIEW toolbar.

While the run is proceeding, the FullBufferAvailable gauge tells that the data taking is on and the depth of the Pool Buffers. It should hover between 15 buffers and 0 buffers if the system is keeping up. If you grab the window and drag it around with the left mouse button pressed, you hold off data taking and the buffer can fill up deeper, and will then bleed off slowly.

The Stop button stops the acquisition.

5.1.2 Inside the Scope Demo

In scope.vi, the first VI is ADC64_Open.vi. This VI downloads COFF executable for target and initializes both Ventura DLL and LabVIEW driver DLL. This is followed by call to ADC64_ConfigAcq.vi which takes values from front panel settings: MuxDepth, AdcCount and DataRate that are used to configure an acquisition.

Next, StartRecordMode.vi is called which starts the target taking data which is queued up on the target but not passed up to host until get External Trigger or TriggerRecordMode.vi is called.

The main loop of the acquisition is in sequence step 3, where AdcBufferAvailable.vi is repeatedly called until it returns nonzero which indicates enough data is available for user to use from within LabVIEW. This copy from LabVIEW driver to LabVIEW is done in GrabAdcBuffer.vi call. Note: maximum size of LabVIEW buffer is 0x2000 (8192) samples. The LabVIEW driver is most efficient when the LabVIEW buffer size is maximized.

When user is done with acquisition pressing STOP button on front panel will stop acquiring and call ADC64_Close.vi which will cleanup all resources used by the LabVIEW driver and the Ventura DLL.

5.2 SigGen

5.2.1 Running the SigGen Demo

The SigGen.vi example provides a simple dual channel signal generator on the 2 DACs of the ADC64. There are 3 waveforms available: DC, sine and square wave. The signals can be viewed by an oscilloscope.

5.2.2 Inside the SigGen Demo

In the file SigGen.vi, the first function is ADC64_Open.vi. It opens up the system much as in the Scope demo. Next, ADC64_ConfigPlay.vi is called which sets up DataRate and which DACs are active.

On the target, when the run starts the program attempts to prefill its queue as much as possible with data. Since we are polling and may not be responsive, we use sequence step 2 to prefill our output pool with data by calling DacBufferAvailable.vi coupled with GrabDacBuffer.vi before calling StartPlaybackMode.vi until we have no empty buffers left. This allows us to prefill at least 16 buffers into the pool.

Then, we call StartPlaybackMode.vi which starts sending data from Ventura to target to be played out the DACs.

In this example, all data we send will be output as we send it. The data sent was carefully arranged to match the format expected by the analog interrupt based on the run configuration we sent to it.

The main loop of playback uses DacBufferAvailable.vi and GrabDacBuffer.vi to take LabVIEW buffers and send them to Ventura to be played out the DACs.

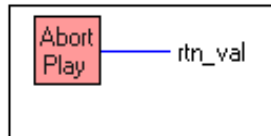
Finally, we call ADC64_Close.vi to clean up resources and close the LabVIEW driver.

6. LabVIEW Driver VIs

AbortPlaybackMode

Halts a running Playback session. AbortPlaybackMode.vi sends a message to the ADC64 target that will stop a currently running Playback session as soon as the command arrives.

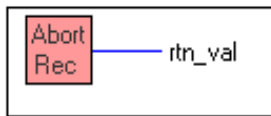
This function should only be called in playback mode.



AbortRecordMode

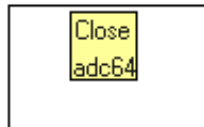
Halts a running Recording session. AbortRecordMode.vi sends a message to the ADC64 target that will stop a currently running Record session as soon as the command arrives.

If not in recording mode, this function has no effect.



ADC64_Close

Shuts Ventura system down and cleans up resources used by the LabVIEW driver. This VI needs to be called when finished using the LabVIEW driver.

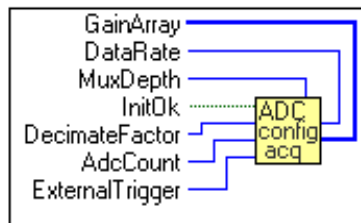


ADC64_ConfigAcq

Set up target for acquisition. ADC64_ConfigAcq.vi transfers the parameters for a Recording to the target. This function may not be sent while in Record or Playback mode.

This function is the first in the setup sequence for the Record mode, being followed by StartRecordMode.vi and TriggerRecordMode.vi.

DataRate is sample rate seen by muxed A/D and is in Hz. The allowed values of MuxDepth are from 1 to 8. A value of 1 indicates no muxing. The allowed values of AdcCount are from 1 to 8. The allowed values of DecimateFactor are from 1 to 128. The allowed values for each GainArray entry are from 0 to 3. There are 8 elements in GainArray corresponding to 8 A/Ds on the ADC64. The InitOk input is used to shut down the application if ADC64_Open.vi fails. The allowed values of ExternalTrigger are 0 for false and 1 for true.

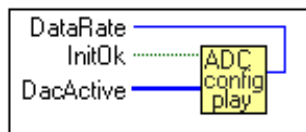


ADC64_ConfigPlay

Configures target for a playback out to DACs. DataRate is rate at which DACs will play out data. DacActive is an array of which DAC channels will output data. The allowed values for each DacActive entry are 0 (inactive) and 1 (active). There are 2 elements in DacActive array corresponding to the 2 DACs on the ADC64. The InitOk input is used to shut down the application if ADC64_Open.vi fails.

This function is the first in the setup sequence for the playback mode, being followed by StartPlaybackMode.vi.

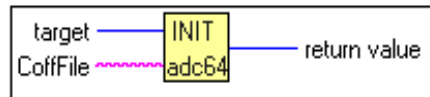
This function may not be sent while in Record or Playback mode.



ADC64_Init

ADC64_Init.vi downloads COFF executable to selected target, sets up Ventura environment and initializes the LabVIEW driver for this run. The `CoffFile` is the Ventura target executable that communicates with the host Ventura DLL.

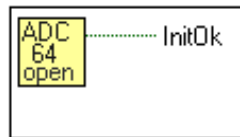
Don't call ADC64_Init.vi directly, call ADC64_Open.vi instead. ADC64_Open.vi calls this lower-level VI.



ADC64_Open

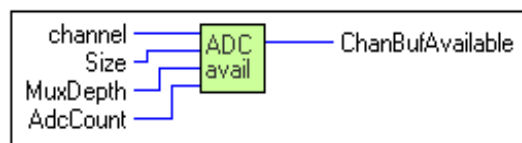
Opens the Ventura system and initializes the target for use with LabVIEW. The `InitOk` output indicates that the LabVIEW driver initialized successfully.

This VI needs to be called before calling any other VIs in the LabVIEW driver.



AdcBufferAvailable

Polls LabVIEW driver to see if requested buffer is available from an acquisition in progress. `MuxDepth` and `AdcCount` determine which channels are active. The allowed values of `MuxDepth` are from 1 to 8. A value of 1 indicates no muxing. The allowed values of `AdcCount` are from 1 to 8. Size of buffers is limited to 0x2000 (8192) 16-bit values (shorts). Any buffers greater than 0x2000 will be truncated. The `channel` input uses internal queue to check to see if has amount of data requested by `Size`. Note: Need to check an active `channel`. A `ChanBufAvailable` return value of 1 indicates that have acquired `Size` amount of data and can now proceed to get it with `GrabAdcBuffer.vi`



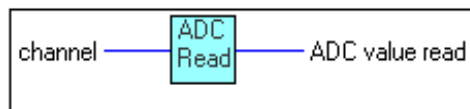
AdcRead

Asynchronous read of an ADC channel. `AdcRead.vi` sends a message to the ADC64 that retrieves data from an ADC channel. The `channel` is the ADC to read, from 0 to 7.

The function returns the signed result of the last ADC conversion. It has 16 bits of significance.

This function is intended for low bandwidth data taking, as the command channel can only be transmitted at rates of about 2 kHz, even if no other processing is taking place on the ADC64.

The use of this function while using the Record Mode streaming on the same channel may disrupt the system and should be avoided. Otherwise, this command can be used while streaming modes are on.

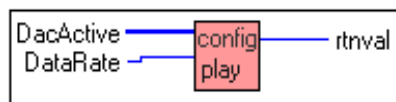


ConfigPlaybackMode

Set up target for Playback mode. `ConfigPlaybackMode.vi` transfers the parameters for Playback to the target. `DataRate` is rate at which DACs will play out data. `DacActive` is an array of which DAC channels will output data. The allowed values for each `DacActive` entry are 0 (inactive) and 1 (active). There are 2 elements in `DacActive` array corresponding to the 2 DACs on the ADC64.

This function may not be sent while in Record or Playback mode.

Don't call `ConfigPlaybackMode.vi` directly, call `ADC64_ConfigPlay.vi` instead. `ADC64_ConfigPlay.vi` calls this lower level VI.



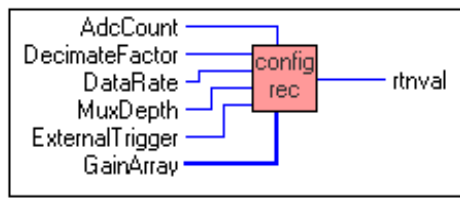
ConfigRecMode

Set up target for acquisition. ConfigRecMode.vi transfers the parameters for a Recording to the target.

DataRate is sample rate seen by muxed A/D and is in Hz. The allowed values of MuxDepth are from 1 to 8. A value of 1 indicates no muxing. Note: as change MuxDepth, need to change DataRate because Ventura assumes actual data rate is desired data rate for that muxed channel. For example, if MuxDepth = 2, then set DataRate to 100 kHz which yields actual sampling rate of 200 kHz to the A/D, but only 100 kHz to each muxed input. The allowed values of AdcCount are from 1 to 8. The allowed values of DecimateFactor are from 1 to 128. The allowed values for each GainArray entry are from 0 to 3. There are 8 elements in GainArray corresponding to 8 A/Ds on the ADC64. The InitOk input is used to shut down the application if ADC64_Open.vi fails. The allowed values of ExternalTrigger are 0 for false and 1 for true.

This function may not be sent while in Record or Playback mode.

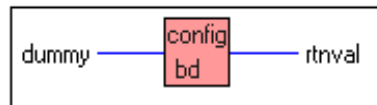
Don't call ConfigRecMode.vi directly, call ADC64_ConfigAcq.vi instead. ADC64_ConfigAcq.vi calls this lower level VI.



ConfigureBoard

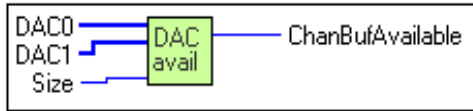
Send board configuration to the target. ConfigureBoard.vi sends the preliminary Board Configuration command to the target, which carries information about the configuration of the target board that cannot be detected by the target. This function may not be sent while in Record or Playback mode.

In this release, there is no data required by the user. However, this function also sends the Pool Buffer size to the target using this command, so this command must be sent before entering Playback or Record mode.



DacBufferAvailable

This VI checks to see if have room in internal playback queue to add more data from LabVIEW to play out to DACs. DAC0 is a LabVIEW buffer that will be combined with DAC1 LabVIEW buffer, then added to an internal playback queue that is transferred to Ventura. Once data is transferred to Ventura, Ventura controls transfer of data to the target. Size of buffers is limited to 0x2000 (8192) 16-bit values (shorts). Any buffers greater than 0x2000 will be truncated. A ChanBufAvailable return value of 1 (true) is used to indicate if have room in internal queue for more data from LabVIEW.

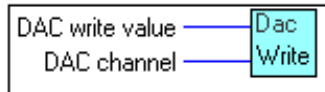


DacWrite

Asynchronous write to a DAC channel. DacWrite.vi sends a message to the ADC64 that sends data to a DAC channel for output. DAC channel is the DAC to change output, from 0 to 1.

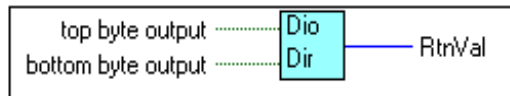
This function is intended for low bandwidth data output, as the command channel can only be transmitted at rates of about 2 kHz, even if no other processing is taking place on the ADC64.

This function cannot be used while Playback Mode streaming is in process, as the DACs are both in use. Otherwise, this command can be used while streaming modes on are on.



DioDirection

Asynchronous digital I/O configuration. DioDirection.vi sends a message to the ADC64 that configures the direction of the bytes of the Digital I/O on the ADC64. A value of 1 configures the byte as output, and 0 as input. The ADC64 only has 16 bits of Digital I/O, so top byte output controls upper byte and bottom byte output controls lower byte.

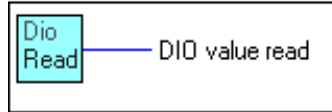


DioRead

Asynchronous digital I/O read. DioRead.vi returns the values of the digital I/O bits on the ADC64. The ADC64 only has 16 bits of digital I/O, so only the lower 16 bits are significant.

This function is intended for low bandwidth data taking, as the command channel can only be transmitted at rates of about 2 kHz, even if no other processing is taking place on the ADC64.

This command can be used while streaming modes are on.

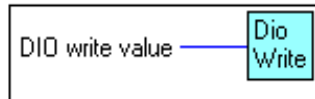


DioWrite

Asynchronous digital I/O write. DioWrite.vi sets the values of the digital I/O bits on the ADC64 to match the bit pattern in DIO write value. The ADC64 only has 16 bits of digital I/O, so only the lower 16 bits are significant.

This function is intended for low bandwidth data, as the command channel can only be transmitted at rates of about 2 kHz, even if no other processing is taking place on the ADC64.

This command can be used while streaming modes are on.

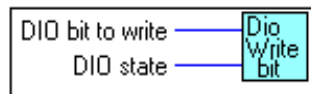


DioWriteBit

Asynchronous digital I/O bitwise write. DioWriteBit.vi sets the value of a single digital I/O bit on the ADC64 to match the value of DIO state. The argument DIO bit to write tells which of the 16 bits of digital I/O is to be set, ranging from 0 to 15.

This function is intended for low bandwidth data, as the command channel can only be transmitted at rates of about 2 kHz, even if no other processing is taking place on the ADC64.

This command can be used while streaming modes are on.



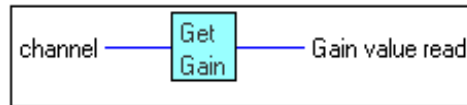
GetGain

Get current ADC gain setting. GetGain.vi reads the gain setting for an ADC on the ADC64 given by channel1. The allowed values of channel are from 0-7.

The function returns a “gain mode” index from 0 to 3, to support both binary (x1, x2, x4, x8) and decimal (x1, x2, x5, x10) gain amplifiers without change.

This function is intended for low bandwidth data, as the command channel can only be transmitted at rates of about 2 kHz, even if no other processing is taking place on the ADC64.

This command can be used while streaming modes are on.



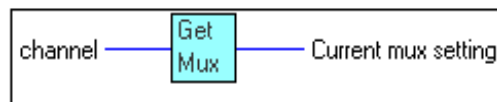
GetMux

Get ADC mux setting. GetMux.vi reads the mux setting for an ADC on the ADC64 given by channel1. The allowed values of channel are from 0-7.

The function returns the current Mux setting from a shadow register which can vary from 0-7 for single ended ADC64s or from 0-3 for differential ADC64s. This shadow register is not updated by the Automatic multiplexing features of the ADC64, so the actual setting may not match if a recording run has intervened from when the last SetMux.vi was called.

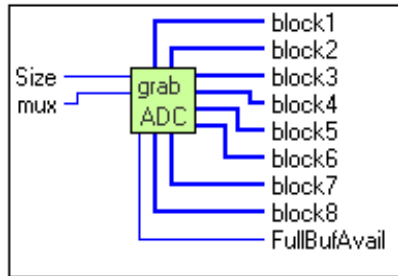
This function is intended for low bandwidth data, as the command channel can only be transmitted at rates of about 2 kHz, even if no other processing is taking place on the ADC64.

This command can be used while streaming modes are on, but if recording the Automatic Multiplexing feature may make its results meaningless.



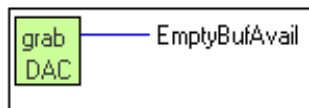
GrabAdcBuffer

This function is called after `AdcBufferAvailable.vi` returns a 1 indicating that there is data available. `GrabAdcBuffer.vi` copies the data from the LabVIEW driver to all LabVIEW buffers for an indicated mux input. For example, if `mux = 1` (`MuxDepth = 1`), then `block1 = IN0 (ADC0)`, `block2 = IN8 (ADC1)`, `block3 = IN16 (ADC2)`, etc. which gives you all 8 Adc inputs with no muxing. The LabVIEW buffers (`block1` through `block8`) are created by LabVIEW to size determined by `Size` and passed to LabVIEW driver. The `FullBufAvail` output is used as rough indicator to see if application is keeping up. If `FullBufAvail` equals 16 then probably have not kept up and there will be gaps in the data.



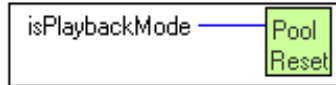
GrabDacBuffer

This function is called after `DacBufferAvailable.vi` returns a 0 indicating that there is no room in playback queue. `GrabDacBuffer.vi` has Ventura extract packed data from the LabVIEW driver to the target. Once data is transferred to the target it is played out the DACs. The return value of `EmptyBufAvail` gives rough indication to user of how well Ventura is keeping up with LabVIEW environment. Since DACs have to play out a fixed rate, the host has to wait until target has played out data before the target can accept more.



PoolReset

Clear Pool variables for a new run. PoolReset.vi reinitializes the pool and its associated variables to a default state. This state is a pool with all buffers empty of data and ready for filling. This function should be called before each Record or Playback run, to avoid any mixing of the data from the previous run. This function is called inside of ADC64_ConfigAcq.vi and ADC64_ConfigPlay.vi so it doesn't need to be called directly. Set isPlaybackMode to 1 for a Playback run, or 0 for Record run.



PoolStop

Release all pending Pool waits. PoolStop.vi is used by a monitor thread to stop all threads that are waiting on pool buffers and return them to a ready state. Normally, this function does not need to be called, since if a run terminates on its own no thread is waiting, and the mode abort functions each perform a PoolStop internally.

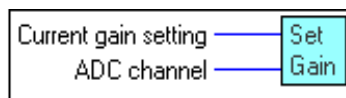


SetGain

Set current ADC gain setting. SetGain.vi changes the gain setting for an ADC on the ADC64 given by ADC channel. The allowed values of ADC channel are from 0-7. The Current gain setting argument is a "gain mode" index from 0 to 3, to support both binary (x1, x2, x4, x8) and decimal (x1, x2, x5, x10) gain amplifiers without change.

This function is intended for low bandwidth data, as the command channel can only be transmitted at rates of about 2 kHz, even if no other processing is taking place on the ADC64.

This command can be used while streaming modes are on.

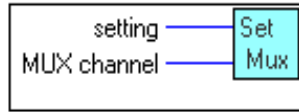


SetMux

Set current ADC mux setting. SetMux.vi changes the mux setting for an ADC on the ADC64 given by channel. The allowed values of MUX channel are from 0-7. The setting argument is the mux channel, which can vary from 0-7 for a single-ended ADC64 or from 0-3 for a differential ADC64.

This function is intended for low bandwidth data, as the command channel can only be transmitted at rates of about 2 kHz, even if no other processing is taking place on the ADC64.

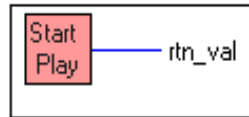
This command can be used while streaming modes are on, but if Automatic multiplexing is in progress in Record mode the true channel may not match the expected one.



StartPlaybackMode

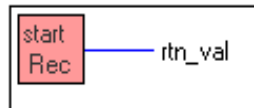
Begin Playback mode. StartPlaybackMode.vi begins the processing of the Playback mode. After this call buffers will begin to be requested and processed.

Playback tries to prefetch enough buffers to fill its internal queue before beginning to output data, so sufficient data should be available to satisfy these requests.



StartRecordMode

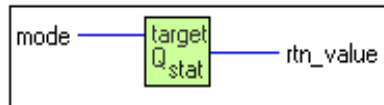
Begin Record mode. StartRecordMode.vi begins the analog interrupt for the Recording mode initiated by a ADC64_ConfigAcq.vi, but actual transferring of buffers does not begin until either an external hardware start trigger is received on TCLK0 pin of the external connector or TriggerRecordMode.vi is called. From then on data will be busmastered to the host when available.



TargetQueueStatus

Return depth of Target Queue. TargetQueueStatus.vi returns the current depth of the indicated queue on the ADC64 target, in percent full. The mode parameter is 0 (VEN_TARGET_ADCQUEUE) for the ADC Queue or 1 (VEN_TARGET_DACQUEUE) for the DAC Queue.

The primary use of this function is the case where a finite waveform is to be played out and the application needs to detect when the last data point has been sent so that the Abort command will not truncate the output prematurely. After adding the last data buffer to the pool, the application can watch for the moment when the DAC queue empties out and TargetQueueStatus.vi returns 0. This will happen when all data has been output through the DACs.



TriggerRecordMode

Begin Record mode data taking. TriggerRecordMode.vi begins data taking in a Record run initiated by a ADC64_ConfigAcq.vi, and started by StartRecordMode.vi. This function is an alternative way of starting data taking to the external hardware start trigger on the TCLK0 pin of the external connector.

